C Coding Convention

C Program Style Guide

Sanjay Vyas

Overview

This document provides guidelines for C programming coding style and formatting. This is a basic set of universally accepted guidelines for C programming style and would also be application to other programming languages, especially C-Family like C++, Java, C# etc.

The very basic purpose of formatting convention is to increase readability of code per se but also by following a common coding standard, make it easier to teams to read/review code and collaborate better.

Well formatted code not only makes the code readable but also reflects the experience and maturity of the developers, leading companies to use code formatting as one of the metrics in evaluating developers.

Source File Organization

File Header Information

Each source code file must have a comment block header, which states the purpose of that file along with identification information.

```
/**********************************************************
* Program: MagicSquare.c
* Author: Sanjay Vyas (SV)
*
* Description:
* Magic Square generates a square where all rows, columns
* and diagonals produce the same sum.
*
* Revision History:
* 1-Jan-2015 (SV): Project created.
* 2-Jan-2015 (SV): Added basic logic for square.
* 3-Jan-2015 (SV): Bug Fix: Invalid input caught correctly.
**********************************************************/
```

Progam is the name of the physical file
Author is the name(s) of the programmer(s)
Description should give an overall synopsis of what the program does
Revision History tracks changes made to the file along with author name

#include Section

The #includes should be right below the file header information and should follow the order

1. System header files
2. User defined header files

```
// Include header files
#include <stdio.h>
#include <stdlib.h>

#include "MyMacros.h"
```

- There should NOT be a space between # and include.
- There SHOULD be one space AFTER #include.
- Separate system header files and user header file with one blank line.

THESE ARE INCORRECT

```
// Include header files
#include<stdio.h>     // Keep a space after #include.
# include <stdlib.h>  // Do not keep a space between # and include.
# include"MyMacros.h" // Leave a blank line after system includes.
```

Macro Definations

Macros should be defined near top of the file, just after the Include section.

```
// Macro definitions
#define ROWS (3)
#define COLUMNS (3)
#define sqr(x) ((x)*(x))
```

- There should NOT be a space between # and define.
- There SHOULD be one space AFTER #define.
- Variable-type macros (e.g ROWS) should be in all upper case.
- Function-type macros (e.g. sqr) should be in all lower case.
- Macro parameter should be paranthesised, e.g. (x)
- Entire macro substitution should be paranthesised, e.g ((x)*(x))
- Macros do NOT need a semicolon

THESE ARE INCORRECT

```
// Macro definitions
#defineROWS (3)       // Keep a space after #define.
#define COLUMNS(3)    // Keep a space after macro name.
#define sqr(x) x*x    // Parenthesize macro parameters.
#define max (a, b) a>b?a:b   // No space between macro & param.
#define min(a, b) (a<b?a:b); // Do NOT end macros with a semicolon.
```

Global and Static Variables

Global and static variables should be avoided as much as possible but if used, they should be prefixed properly.

```
// Global and Static Variable
int g_ListCount = 0;
static int s_LastCounter = 0;
```

- Global variables should be prefixed with g_ or global_
- Static variables should be prefixed with s_ or static_

Comments

A program should be well-documented in terms of explaining the workings of the code, stating the purpose of different parts of the code and maintaining revision history.

Multi-line Comments

Multi-line comments in C are enclosed using a pair of `/* and */`.
Multiline comments should be used for program header information, as stated earlier. Multiline comments should also be used for functions headers and explaining logic of piece of code.

FUNCTION HEADER MULTI-LINE COMMENT

```
/************************************************************
* Function: GenerateSquare
* Return Value: void
* Parameters:
*   int size -  An odd value of which the square is to be
*               generated.
*
* Description:  This function takes a positive odd value,
*               allocates memory on heap and then generates
*               a square where all rows, all columns and
*               both diagonal sum up to the same value.
************************************************************/
void GenerateSquare(int size)
```

MULTI-LINE COMMENT EXPLAINING LOGIC

```
// As we are generating a array of any given odd size,
// we need to allocate it on the heap.
// We will first allocate the rows array of pointer with size
// and then allocate each row with size number of columns.
extern int **g_square;
g_square = (int **)malloc(size * sizeof(int *));
if (NULL == g_square)
    return 0;

for (int i=0; i<size; i++)
{
    g_square[i]=(int *)calloc(size, sizeof(int));
    if (NULL == g_square[i])
        return 0;
}
```

Single-line Comments

Single line comments are usually used at the end of a statement. These are typically made using `//.`

Use // for single line comments

```
// Now generate the array.
int row = 0;          // Start at top row.
int col = size/2;     // Start at middle column.

// Fill up the square with values.
for (i=0; i<sqr(size); i++)
{
```

- There should be at least a space before //.
- There should be a exactly one space after //.
- As far as possible, align // on subsequent lines
- Comments are not in C, they are in English, so follow English grammar.

`THESE ARE INCORRECT`

```
int i;// loop var
int value;// value var
//do the alloc
```

Identifier naming convention

Identifiers for variables, constants, macros, etc. should be named in a meaningful manner. Very short names should be avoided, e.g., aaa, abc, n, m, l.
Identifiers, while being crisp, should reflect the purpose of the variable or function.

Identifier Names

```
int GenerateSquare(int size)
{
    extern int **g_square;
    int i;      // Loop variable for memory allocation.
    int value;  // Value variable for filling up the square.
```

- Indetifier name should indicate the purpose.
- Avoid very short names, except in well known cases like i and j.

THESE ARE INCORRECT

```
int fun(int a, const char * v[])
{
    int aaa;
    int pnm;
    int l,m,n;
```

Indentation

Code should be well indented to make it more readable. Consistent indentation should be followed thruout the code and the indentations should be made using the tab key instead of multiple spaces.

Indenting blocks

INDENTING NESTED STATEMENTS

```
if (NULL == g_square)
    return 0;
for (int i=0; i<size; i++)
{
    g_square[i]=(int *)calloc(size, sizeof(int));
    if (NULL == g_square[i])
        return 0;
}
```

- Blocks { } should be aligned one below the other.
- For every level, indent with one TAB (4 is usually the standard).

THESE ARE INCORRECT

```
for (value=0; value<sqr(size); value++)
{
g_square[row][col]=value;
    row--;
col++;
if (row < 0 && col>=size)
{
row=size-1;
col=0;
} else
    if (row <0) row=size-1;
    else
    if (col == size)
    col=0;
```

Spacing

Spacing is extremely important in making a program readable and should be used consistently thru-out the code.

Spacing after keywords

The general rule is to leave exactly one space after any keyword, which includes keywords if, for, while, else, etc.

EXACTLY ONE SPACE AFTER KEYWORDS

```
for (int i=0; i<MAX; i++)  // One space after keyword for.
{
    if (i%2)               // One space after keyword if.
        sum += i;
}
```

Spacing in function calls

To differentiate function calls from keywords, no space should be left after the function name, either in definition or in function call.

```
int max(int a, int b)           // No space after max.
{
    return a>b?a:b;             // One space after return
}

int main()                      // No space after main.
{
    printf(%d\n", max(5, 7));   // No space after printf/max.
    return 0;                  // One space after return.
}
```

Spacing around operators

While we program in C, follow basic rules of English language for punctuation marks. Hence, there should be a single space after punctuations marks like , ; :

```
for (value=0; value<sqr(size); value++) // Space after ;
{


int c = max(5, 7);         // Space after ,

if (row < 0 && col >= size)// Space around && < >=
```